Contents

SynchroAPI Reference

API version	v1
API path prefix	/api/v1/
Documented against	SynchroServer 2.7.9.055
Last updated	2026-09-02

REST API for SynchroServer. Base URL: `http://<host>:8080/api/v1/` (or `https://<host>:8443/api/v1/` when HTTPS is enabled — see the User Guide for HTTPS configuration). The `curl` examples below use plain HTTP; substitute `https` and the configured `HSPORT` when applicable.

The `v1` segment in the URL **is** the API major version. Backwards-incompatible changes go to `/api/v2/` and run side-by-side with `v1`. Additive changes (new endpoints, new optional fields in responses) ship under `v1` without bumping the major. The current SynchroServer build version is reported by `GET /health` — clients that need to gate features on a specific server build should read its `version` field rather than parsing the URL.

Most endpoints require the header `Authorization: Bearer <TOKEN>` (see Authentication). A small number of routes are explicitly public — see Public (no-auth) endpoints below.

Authentication

Login

```
curl -X POST http://localhost:8080/api/v1/auth/login \
  -H 'Content-Type: application/json' \
  -d '{"username":"synchro","password":"synchro"}'
```

Response:

```
{
  "token": "eyJhbGciOiJIUzI1NiJ9...",
  "login_id": "0",
  "expires_in": 28800
}
```

Store the token for subsequent requests:

```
TOKEN=$(curl -s -X POST http://localhost:8080/api/v1/auth/login \
  -H 'Content-Type: application/json' \
  -d '{"username":"synchro","password":"synchro"}' | jq -r '.token')
```

Rate limiting

Failed logins are rate-limited per client IP. Successful logins are never delayed and clear the caller's history.

Condition	Response
5 failures within 60s	`429` until the window drains
10 failures within 900s	`429` for 900s

```
{"error":"too many failed login attempts","code":429,"retry_after":42}
```

A `Retry-After` header carries the same value in seconds. Tunable with `ApiLoginThrottle`, `ApiLoginThrottleLimits` and `ApiTrustProxyHeaders` in `server.ini` — see `PasswordManagement-<version>.md` §7.

POST /auth/password

Change the **calling user's own** password. Not admin-gated: any authenticated user may rotate their own credentials.

```
curl -s -X POST http://localhost:8080/api/v1/auth/password \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"old_password":"synchro","new_password":"a-new-secret"}'
```

Response:

```
{"status":"password changed"}
```

Status	Meaning
`200`	Password changed; the stored value is now a PBKDF2 hash
`400`	Missing field, or the new password fails policy
`403`	`old_password` does not match
`404`	The account no longer exists
`429`	Too many failed attempts — see `Retry-After`

Password policy (rejected with `400`): empty or whitespace-only, shorter than `ApiPasswordMinLength` (default 8), containing `:`, equal to the username, equal to the current password, or one of the shipped default passwords. The same policy applies to the admin routes below.

Existing sessions stay valid after a change; the JWT is not invalidated.

> Prefer this over the legacy `admin.pl?Action=changeLoginPassword` GWT call, which passes the old and new passwords as URL query parameters and therefore writes them to access logs, proxy logs and browser history.

GET /auth/me

Identity and capabilities of the token holder.

```
curl -s http://localhost:8080/api/v1/auth/me -H "Authorization: Bearer $TOKEN"
```

```
{
  "login_id": "0",
  "username": "synchro",
  "fullname": "Synchro",
  "is_admin": true,
  "is_readonly": false,
  "password_is_plaintext": false,
  "password_min_length": 8
}
```

`password_is_plaintext` reports whether this account's stored password has been converted to a hash yet. It exposes no part of the password itself; SynchroUI uses it to prompt for a rotation.

Cross-origin requests (CORS)

By default the API sends **no** `Access-Control-*` headers. SynchroUI is served by the same SynchroServer instance, so it is same-origin and needs none, and non-browser clients — `curl`, this test suite, SynchroClient — ignore CORS entirely.

If you serve a browser client from a different origin, list it:

```
# conf/server-local.ini
ApiAllowedOrigins=https://dash.example.com,http://localhost:5173
```

The request's `Origin` is echoed back only on an exact match, together with `Vary: Origin`. No `Access-Control-Allow-Credentials` is sent — the API authenticates with a Bearer token rather than cookies.

> **Changed in 2.7.9.** The API previously sent `Access-Control-Allow-Origin: *` unconditionally, which let any website a logged-in user visited issue authenticated `GET`, `POST`, `PUT` and `DELETE` calls on their behalf (vulnerability report finding 4.1).
>
> Setting `ApiAllowedOrigins=*` restores the old behaviour and logs a warning at startup. It re-opens the finding.

Health

GET /health

Liveness check. No authentication required.

```
curl http://localhost:8080/api/v1/health
```

Response:

```
{"status":"ok","version":"SynchroServer-2.7.8.048"}
```

The `version` field is the SynchroServer build version (loaded from `ca/synchro/server/version.properties` inside `SynchroServer.jar` at startup). Use this for build-level feature detection. The API major version is encoded in the URL prefix (`v1`), not here.

Public (no-auth) endpoints

A small set of routes are intentionally exposed without authentication so they can be embedded in dashboards, kiosks, or third-party tools. They mirror their authenticated counterparts but are read-only.

Public route	Authenticated equivalent	Notes
`GET /public/index`	`GET /index`	Same query params, same response shape
`GET /public/data/query`	`GET /data/query`	Same query params, same response shape
`GET /public/data/timeseries`	`GET /data/timeseries`	Same query params, same response shape
`GET /public/periods`	`GET /periods`	Same response shape
`GET /public/views/{user}/{name}`	`GET /views/{user}/{name}`	**Returns 403 unless the view's `sharing` field is `"public"`**
`GET /download/{filename}?token=...`	(none)	Streams a release file using a signed token issued by `POST /admin/downloads`. See Admin: Releases & Downloads.

Example:

```
# No Authorization header needed
curl http://localhost:8080/api/v1/public/data/query \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d 'function=LAST' -d 'limit=1'
```

> ⚠️ Operators should put public endpoints behind a network ACL or reverse-proxy rate-limit if the SynchroServer is reachable from the open internet, since they perform DQM queries against your time series store.

Time Series Index

GET /index

Browse the time series index with optional filters (regex supported).

```
# Full index scan
curl http://localhost:8080/api/v1/index \
  -H "Authorization: Bearer $TOKEN"

# Filtered by component and type
```

```
curl http://localhost:8080/api/v1/index \
  -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST'

# Filtered by metric with regex
curl http://localhost:8080/api/v1/index \
  -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'metric=CPU|MEM'
```

Parameters: | Param | Default | Description | |——-|———|————-| | `component` | `.*` | Component filter (regex) | | `type` | `.*` | Type filter (regex) | | `metric` | `.*` | Metric filter (regex) | | `target` | `.*` | Target filter (regex) | | `date` | `D-00` | Date filter | | `max_load_count` | `1000` | Max results |

Response:

```
{
  "ts_count": 26,
  "component": ["affa5"],
  "type": ["HOST", "PROC"],
  "metric": ["CPU", "MEM", "WAIT"],
  "target": ["HOST", "SynchroServer.jar"]
}
```

Data Query

GET /data/query

Query time series data with cursor-based pagination.

```
# Basic query - last value for CPU on HOST
curl http://localhost:8080/api/v1/data/query \
  -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d 'function=LAST' -d 'limit=5'

# All functions at once
curl http://localhost:8080/api/v1/data/query \
  -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d 'function=ALL'

# With pagination cursor
curl http://localhost:8080/api/v1/data/query \
  -H "Authorization: Bearer $TOKEN" \
```

```
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=.*' -d 'target=.*' \
  -d 'limit=10' -d 'cursor=<base64_cursor>'
```

Parameters: | Param | Default | Description | |———-|———|————-| | `component` | `.*` | Component filter (regex) | | `type` | `.*` | Type filter (regex) | | `metric` | `.*` | Metric filter (regex) | | `target` | `.*` | Target filter (regex) | | `date` | `D-00` | Date (`D-00` = today, `D-01` = yesterday, etc.) | | `period` | (all) | Period filter (e.g., `00@24`) | | `function` | `Last` | `Last`, `Min`, `Max`, `Average`, `Sum`, `Count`, `ALL` | | `sort` | `Last` | Sort field | | `limit` | `100` | Results per page (max 1000) | | `cursor` | (none) | Pagination cursor from previous response |

Response:

```
{
  "data": {
    "Count": 1,
    "ts": [
      {
        "component": "affa5",
        "type": "HOST",
        "metric": "CPU",
        "target": "HOST",
        "date": "20260330",
        "period": "00@24",
        "last": 12.5
      }
    ]
  },
  "pagination": {
    "limit": 5,
    "offset": 0,
    "has_more": false,
    "cursor": null
  }
}
```

Time Series (Charting)

GET /data/timeseries

Bucketed time series data optimized for charting. Returns aligned timestamp and value arrays.

```
# Last hour, 300 points
curl http://localhost:8080/api/v1/data/timeseries \
  -H "Authorization: Bearer $TOKEN" \
```

```
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d "from=$(($(date +%s) - 3600))" -d "to=$(date +%s)" \
  -d 'max_points=300' -d 'agg=avg' -d 'function=Last'

# Last 24 hours, 1-second resolution (up to 10000 points)
curl http://localhost:8080/api/v1/data/timeseries \
  -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d "from=$(($(date +%s) - 86400))" -d "to=$(date +%s)" \
  -d 'max_points=10000'
```

Parameters: | Param | Default | Description | |——-|———|————-| | `component` | (required) | Exact component name | | `type` | (required) | Exact type name | | `metric` | (required) | Exact metric name | | `target` | (required) | Exact target name | | `from` | `now - 3600` | Start time (Unix epoch seconds) | | `to` | `now` | End time (Unix epoch seconds) | | `max_points` | `1000` | Max data points (capped at 10000) | | `agg` | `avg` | Bucketing aggregation: `avg`, `max`, `min`, `last`, `sum` | | `function` | `Last` | DQM stat function |

Response:

```
{
  "meta": {
    "component": "affa5",
    "type": "HOST",
    "metric": "CPU",
    "target": "HOST"
  },
  "from": 1774830000,
  "to": 1774833600,
  "bucket_seconds": 12,
  "point_count": 300,
  "t": [1774830012, 1774830024, ...],
  "v": [15.2, 14.8, ...]
}
```

`bucket_seconds=1` means raw (no bucketing). The server computes: `bucket_seconds = ceil(time_range / max_points)`

Periods

GET /periods

List defined time periods.

```
curl http://localhost:8080/api/v1/periods \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "periods": [
    {"name": "00@24", "start": "000000", "end": "240000"},
    {"name": "08@17", "start": "080000", "end": "170000"}
  ]
}
```

Views

Views store SynchroUI dashboard configurations as JSON. Named as `username/viewname`.

GET /views

List all accessible views (own + shared + public).

```
curl http://localhost:8080/api/v1/views \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "views": [
    {"name": "synchro/Dashboard", "owner": "synchro", "own": true},
    {"name": "admin/SharedView", "owner": "admin", "own": false}
  ]
}
```

GET /views/{user}/{name}

Get a view's JSON content.

```
curl http://localhost:8080/api/v1/views/synchro/Dashboard \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "name": "synchro/Dashboard",
  "owner": "synchro",
  "view": {
    "panels": [
```

```
      {
        "title": "CPU Usage",
        "selections": [
          {"component": "affa5", "type": "HOST", "metric": "CPU", "target":
"HOST"}
        ],
        "activeTab": "graph",
        "queryFunctions": ["Last"],
        "activePresetSeconds": 3600,
        "refreshInterval": 5000,
        "selectedPeriods": ["00@24"],
        "resolution": 0,
        "size": "L"
      }
    ],
    "sharing": "public"
  }
}
```

POST /views/{user}/{name}

Create or update a view.

```
curl -X POST http://localhost:8080/api/v1/views/synchro/MyView \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{
    "panels": [
      {
        "title": "Server CPU",
        "selections": [
          {"component": "affa5", "type": "HOST", "metric": "CPU", "target":
"HOST"}
        ],
        "activeTab": "graph",
        "queryFunctions": ["Last"],
        "activePresetSeconds": 3600,
        "refreshInterval": 5000,
        "selectedPeriods": ["00@24"],
        "size": "L"
      }
    ],
    "sharing": "public"
  }'
```

Sharing options: - `"sharing": null` — private (owner only) - `"sharing": "public"` — visible to all users - `"sharing": ["user1", "user2"]` — shared with specific users

DELETE /views/{user}/{name}

Delete a view (owner or admin only).

```
curl -X DELETE http://localhost:8080/api/v1/views/synchro/MyView \
  -H "Authorization: Bearer $TOKEN"
```

Alarms

GET /alarms/state

Get alarm status and all alarm definitions (any authenticated user).

```
curl http://localhost:8080/api/v1/alarms/state \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "enabled": true,
  "alarms": [
    {
      "Name": "HOST-CPU",
      "Active": true,
      "AlarmsItems": [
        {
          "Description": "CPU usage alarm",
          "Component": ".*",
          "Type": "HOST",
          "Metric": "CPU",
          "Target": "HOST",
          "Date": ".*",
          "Period": ".*",
          "Function": "LAST",
          "OKMin": 0, "OKMax": 50,
          "WarnMin": 50, "WarnMax": 75,
          "CriticalMin": 75, "CriticalMax": 90,
          "FatalMin": 90, "FatalMax": 100,
          "RaiseCriteria": "BOTH",
          "RaiseLevel": "OK",
          "Exemption": 0,
          "UseByTime": false,
          "Delay": 0,
          "Snooze": 0,
          "Repeat": 0
        }
      ]
    }
```

```
  ]
}
```

Alarm levels: 0=OK, 1=Warning, 2=Critical, 3=Fatal. When an alarm triggers, a sister TS is created with `-ALARM` appended to the Type.

Admin: Alarms

GET /admin/alarms

Get full alarm configuration (admin only).

```
curl http://localhost:8080/api/v1/admin/alarms \
  -H "Authorization: Bearer $TOKEN"
```

POST /admin/alarms

Save alarm configuration in JSON format.

Storage layout (since 2.7.8.048): the request body's `Alarms` array is **split into one file per group**. For each group object, the server writes:

```
conf/alarm-json.d/api-alarms-${SafeName}.json
```

…where `SafeName` is the group's `Name` field with any character outside `[A-Za-z0-9._-]` replaced by `_`. Each per-group file uses the same envelope shape as the request body, wrapping just that one group:

```
{ "Alarms": [ { "Name": "DEFAULT", "Active": true, "AlarmsItems": [ … ] } ] }
```

Cleanup: any existing `api-alarms*.json` file in `alarm-json.d/` that is not in the new request is deleted on save. This means: - Removing a group via the UI removes its file from disk on the next save. - The legacy combined `api-alarms.json` (from before 2.7.8.048) is removed automatically the first time you save through the API.

Validation errors (HTTP 400): - A group is missing a non-empty `Name` field. - Two groups would resolve to the same `SafeName` after sanitization (e.g. `My Alarms` and `My/Alarms` both become `My_Alarms`).

After a successful save the in-memory `AlarmManager` is reloaded, so changes take effect immediately.

```
curl -X POST http://localhost:8080/api/v1/admin/alarms \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{
    "Alarms": [
      {
        "Name": "CPU-HIGH",
        "Active": true,
        "AlarmsItems": [
          {
            "Description": "High CPU usage",
            "Component": ".*",
            "Type": "HOST",
            "Metric": "CPU",
            "Target": "HOST",
            "Date": ".*",
            "Period": ".*",
            "Function": "LAST",
            "OKMin": 0, "OKMax": 50,
            "WarnMin": 50, "WarnMax": 75,
            "CriticalMin": 75, "CriticalMax": 90,
            "FatalMin": 90, "FatalMax": 100,
            "RaiseCriteria": "BOTH",
            "RaiseLevel": "OK",
            "Exemption": 0,
            "UseByTime": false,
            "Delay": 0,
            "Snooze": 0,
            "Repeat": 0
          }
        ]
      }
    ]
  }'
```

The example above writes one file: `conf/alarm-json.d/api-alarms-CPU-HIGH.json`.

PUT /admin/alarms/enabled

Enable or disable alarms globally.

```
# Disable alarms
curl -X PUT http://localhost:8080/api/v1/admin/alarms/enabled \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"enabled": false}'

# Enable alarms
curl -X PUT http://localhost:8080/api/v1/admin/alarms/enabled \
  -H "Authorization: Bearer $TOKEN" \
```

```
  -H 'Content-Type: application/json' \
  -d '{"enabled": true}'
```

POST /admin/alarms/reload

Reload alarm configuration from disk.

```
curl -X POST http://localhost:8080/api/v1/admin/alarms/reload \
  -H "Authorization: Bearer $TOKEN"
```

Response: `{"status":"reload started"}` (HTTP 202)

PUT /admin/alarms/file/{name} *(since 2.7.9.055)*

Write (create or replace) **one** alarm file, `conf/alarm-json.d/{name}.json`.

Unlike `POST /admin/alarms` — which is a whole-set replace that deletes every `api-alarms*.json` not in the request — this touches **only** the named file. It performs no cleanup of other files, so it cannot disturb `api-alarms.json` (the `DEFAULT` group) or the scheduled-event alarms the event monitor manages (`event-*.json`). This is the right primitive for adding an *individual*, independent alarm file (each file in `alarm-json.d/` is loaded independently).

The `{name}` is sanitized to `[A-Za-z0-9._-]` (any other character becomes _) and the `.json` suffix is added by the server. Two namespaces are **reserved** and rejected with `400`: names that resolve to `api-alarms*` (owned by `POST /admin/alarms`) or `event-*` (owned by the event monitor).

The body is the same `{"Alarms":[<group>, …]}` envelope used by `POST /admin/alarms`; the whole envelope is written to the file verbatim, so a single file may contain one or more groups.

```
curl -X PUT http://localhost:8080/api/v1/admin/alarms/file/my-alarms \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{
    "Alarms": [
      {
        "Name": "CPU-HIGH",
        "Active": true,
        "AlarmsItems": [
          {
            "Description": "High CPU",
            "Component": "web", "Type": "CPU$", "Metric": "USED",
```

```
            "Target": ".*", "Date": ".*", "Period": "00@24",
            "Function": "LAST",
            "RaiseCriteria": "BOTH", "RaiseLevel": "OK",
            "Exemption": 0, "UseByTime": false,
            "Delay": 0, "Snooze": 0, "Repeat": 0,
            "OKMin": 0, "OKMax": 80, "FatalMin": 80, "FatalMax": 100
          }
        ]
      }
    ]
  }'
```

Response: `{"status":"saved","file":"my-alarms.json"}` (HTTP 200). The in-memory `AlarmManager` is reloaded, so the change takes effect immediately.

Errors: `400` (empty body, body missing an `Alarms` array, or a reserved/invalid name), `403` (non-admin token).

DELETE /admin/alarms/file/{name} *(since 2.7.9.055)*

Delete **one** alarm file, `conf/alarm-json.d/{name}.json`. Same name sanitization and reserved-namespace rules as the `PUT`. Deleting an absent file is not an error.

```
curl -X DELETE http://localhost:8080/api/v1/admin/alarms/file/my-alarms \
  -H "Authorization: Bearer $TOKEN"
```

Response: `{"status":"deleted","file":"my-alarms.json"}` (or `"absent"` if the file did not exist), HTTP 200. `AlarmManager` is reloaded after the delete.

Admin: Users

GET /admin/users

List all users.

```
curl http://localhost:8080/api/v1/admin/users \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "users": [
    {
      "uid": "0",
      "username": "synchro",
```

```
      "fullname": "Synchro",
      "gid": "0",
      "reporttime": "010000",
      "is_readonly": false,
      "read_from_users": ""
    }
  ]
}
```

POST /admin/users

Create a new user.

```
curl -X POST http://localhost:8080/api/v1/admin/users \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"username": "newuser", "password": "secret123"}'
```

Response (HTTP 201):

```
{"uid": "ABCDEFGHIJ", "username": "newuser"}
```

PUT /admin/users/{uid}

Update a user.

```
curl -X PUT http://localhost:8080/api/v1/admin/users/ABCDEFGHIJ \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"fullname": "New User", "gid": "1", "is_readonly": false}'
```

DELETE /admin/users/{uid}

Delete a user (cannot delete own account).

```
curl -X DELETE http://localhost:8080/api/v1/admin/users/ABCDEFGHIJ \
  -H "Authorization: Bearer $TOKEN"
```

POST /admin/users/{uid}/password

Reset another user's password. The old password is **not** required — this is the route to use when someone has forgotten theirs.

```
curl -X POST http://localhost:8080/api/v1/admin/users/ABCDEFGHIJ/password \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"password": "newpassword"}'
```

Response:

```
{"status":"password updated"}
```

The new value is stored as a PBKDF2 hash. The user's other fields are untouched, and any token they already hold stays valid until it expires.

Users changing their **own** password should use `POST /auth/password`, which verifies the old one.

> **Password policy applies to all three admin write routes** (`POST /admin/users`, `PUT /admin/users/{uid}` when a `password` field is supplied, and this one). A rejected password returns `400` with the reason: empty or whitespace-only, shorter than `ApiPasswordMinLength` (default 8), containing `:`, equal to the username, or one of the shipped default passwords. On `PUT`, omitting `password` entirely leaves the stored hash untouched.

See `PasswordManagement-<version>.md` for storage format, migration of existing plaintext entries, and how to change the shipped `synchro` default so it survives upgrades.

Admin: Configuration

GET /admin/config/{name}

Read a configuration file.

```
curl http://localhost:8080/api/v1/admin/config/server \
  -H "Authorization: Bearer $TOKEN"
```

Config names: `alarm`, `retention`, `compression`, `aggregator`, `mavg`, `group`, `server`, `client`, `report`, `dropfrom`, `dropmetrics`, `tsmapping`, `valuemapping`

POST /admin/config/{name}

Write a configuration file.

```
curl -X POST http://localhost:8080/api/v1/admin/config/dropmetrics \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"content": "# Drop patterns\n.*_TEST_.*"}'
```

Admin: Jobs

POST /admin/jobs/{name}

Trigger a background job.

```
curl -X POST http://localhost:8080/api/v1/admin/jobs/retention \
  -H "Authorization: Bearer $TOKEN"
```

Job names: `retention`, `log-retention`, `report-retention`, `compression`, `aggregator`, `reports`, `gc`, `load-layouts`, `load-alarms`, `load-reports`, `load-presets`

Response (HTTP 202):

```
{"job": "retention", "status": "started"}
```

Admin: Releases

The Releases / Downloads admin lets operators publish release tarballs (or any file dropped into `data/releases/`) and hand out **signed, time-limited download links** that can be shared without revealing an API token. The download itself is served via the public route `GET /download/{filename}?token=...`.

GET /admin/releases

List files available for sharing. Returns everything in `data/releases/` except dotfiles and `.sha256` companion files, sorted by modification time descending.

```
curl http://localhost:8080/api/v1/admin/releases \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "releases": [
    {
      "name": "SynchroServer-2.7.8.048.tar.gz",
      "size": 45388259,
      "modified": 1775000000,
      "sha256":
"d09e42f0c745e53d0ef4dbaaa65c42385e5684022eae95a619428679d70955ae"
    }
  ]
}
```

`modified` is Unix epoch seconds. `sha256` is the lowercase hex digest of the file's contents (cached on disk in a sibling `.sha256` file so the next listing is fast).

Admin: Downloads

A "download" is a signed token + metadata record stored in `data/releases/.downloads.json`. Each record points at one file in `data/releases/` and carries a TTL, an optional max-download counter, an optional human label, and an audit log of every download attempt.

GET /admin/downloads

List all signed download links (active + revoked + expired).

```
curl http://localhost:8080/api/v1/admin/downloads \
  -H "Authorization: Bearer $TOKEN"
```

Response:

```
{
  "links": [
    {
      "id": "dl-3f9a2c1b8e44",
      "filename": "SynchroServer-2.7.8.048.tar.gz",
      "size": 45388259,
      "sha256": "d09e42f0…",
      "created": 1775003600,
      "expires": 1775608400,
      "max_downloads": 0,
      "download_count": 0,
      "revoked": false,
      "label": "Customer A — Apr release",
      "created_by": "synchro",
      "audit": []
    }
  ]
}
```

POST /admin/downloads

Create a new signed download link. The response includes a JWT-signed `token` that gates access to `GET /download/{filename}` (see below).

```
curl -X POST http://localhost:8080/api/v1/admin/downloads \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
```

```
  -d '{
    "filename": "SynchroServer-2.7.8.048.tar.gz",
    "ttl_days": 7,
    "max_downloads": 0,
    "label": "Customer A — Apr release"
  }'
```

Body fields: | Field | Default | Description | |—|—|—| | `filename` | (required) | Must exist in `data/releases/`. Cannot contain `..` or `/`. | | `ttl_days` | `7` | Token validity in days. Clamped to `[1, 365]`. | | `max_downloads` | `0` | Hard cap on download count. `0` = unlimited. | | `label` | `""` | Free-form human-readable note shown in the listing. |

Response:

```
{
  "id": "dl-3f9a2c1b8e44",
  "token": "eyJhbGciOiJIUzI1NiJ9...",
  "link": { /* same shape as the GET listing entries */ }
}
```

The recipient downloads the file via:

```
GET /api/v1/download/{filename}?token=<token>
```

This route is **public** (no `Authorization` header), but the token is bound to the exact filename, has the configured TTL, and is rejected if the link has been revoked or has hit `max_downloads`. Each successful download appends an entry to the link's `audit` array (timestamp + remote IP).

DELETE /admin/downloads/{id}

Revoke a previously issued link. The record is **kept** in the metadata file (with `"revoked": true`) so the audit history is preserved; subsequent downloads with that token return HTTP 403.

```
curl -X DELETE http://localhost:8080/api/v1/admin/downloads/dl-3f9a2c1b8e44 \
  -H "Authorization: Bearer $TOKEN"
```

Layouts (Legacy)

Legacy layout management for SynchroWEB. See Views for the modern equivalent.

GET /layouts

```
curl http://localhost:8080/api/v1/layouts \
  -H "Authorization: Bearer $TOKEN"
```

GET /layouts/{id}

```
curl http://localhost:8080/api/v1/layouts/default \
  -H "Authorization: Bearer $TOKEN"
```

POST /layouts/{id}

```
curl -X POST http://localhost:8080/api/v1/layouts/MyLayout \
  -H "Authorization: Bearer $TOKEN" \
  -H 'Content-Type: application/json' \
  -d '{"content": "LID=MyLayout\nResultCount=1\n..."}'
```

DELETE /layouts/{id}

```
curl -X DELETE http://localhost:8080/api/v1/layouts/MyLayout \
  -H "Authorization: Bearer $TOKEN"
```

WebSocket Live Push

POST /auth/ws-ticket

Mint a single-use ticket for the WebSocket handshake.

```
TICKET=$(curl -s -X POST http://localhost:8080/api/v1/auth/ws-ticket \
  -H "Authorization: Bearer $TOKEN" | jq -r '.ticket')
```

```
{"ticket":"h1Qw...43-char-url-safe-string...","expires_in":30}
```

A ticket is valid for 30 seconds, works exactly **once**, and grants nothing beyond opening one WebSocket. Each connection needs its own.

WS /ws/subscribe

Subscribe to live time series updates via WebSocket.

```
# 1. Mint a ticket (see above)
TICKET=$(curl -s -X POST http://localhost:8080/api/v1/auth/ws-ticket \
  -H "Authorization: Bearer $TOKEN" | jq -r '.ticket')

# 2. Use it for the upgrade
websocat "ws://localhost:8080/api/v1/ws/subscribe?ticket=$TICKET"
```

```
# Send subscription message
{"action":"subscribe","component":"affa5","ts_type":"HOST","metric":"CPU","target":"HOST"}
```

Status	Meaning
101	Upgraded
401	Missing, invalid, expired or already-used ticket

Changed in 2.7.9. This endpoint previously took the session JWT directly as `?token=<jwt>`. Browsers cannot set an `Authorization` header on a WebSocket upgrade, so the credential has to travel in the URL — where it is written to HTTP access logs, reverse-proxy logs and browser history. That put a long-lived session token in all three (vulnerability report finding 4.3).

A ticket is different in the ways that matter: seconds of life, single use, and useful only for opening a socket. One recovered from a log later is inert.

`?token=` is rejected by default. Setting `ApiAllowWsTokenParam=true` in `server.ini` re-enables it for clients that cannot be updated yet; each use logs a warning, and it re-opens the finding.

Events (Scheduled Event Monitoring)

These endpoints are NOT served by SynchroServer

Changed in 2.7.9. Event monitoring runs as a separate process, `SynchroEventMonitor`, listening on its own port **8090**. SynchroServer itself no longer handles `/api/v1/events*` — an in-JVM implementation existed briefly in an unreleased build, but it froze the server under load and was withdrawn.

Consequences for anything calling these endpoints:

- **`http://localhost:8080/api/v1/events…` now returns 404.** The `localhost:8080` examples below are wrong for a direct call; they work only through a front proxy that routes the path to 8090.

- On live5, Caddy does exactly that: `/api/v1/events` and `/api/v1/events/*` are routed to `localhost:8090`, everything else to SynchroServer on 8080. So `https://live5.synchro-tech.com/api/v1/events` works unchanged.
- **They are currently unauthenticated.** `SynchroEventMonitor` has no auth layer, so the `Authorization: Bearer $TOKEN` header shown in the examples below is accepted but ignored. Do not expose port 8090 directly.
- The service is started independently of SynchroServer and is not managed by `start-server.sh`.

Monitors that the day's *planned* events (reports, application start/stop, phase changes, …) actually happen on time. Event definitions are managed over REST and stored as one JSON file per event under `data/Events/<UniqueID>.json`. A background monitor advances a per-event state machine every second, pushes six metrics into the stat engine over the SC01 protocol, and generates one alarm per event (`conf/alarm-json.d/event-<UniqueID>.json`) so lateness raises the usual Syslog/email alarm sinks. Applications report actual start/completion via the `start` / `complete` ping endpoints.

Emitted time series (per active event planned today, once per second):

Dimension	Value
Component	`<UniqueID>`
Type	`SynchroEvents`
Target	`<EventType>` (Report / AppStart / AppStop / …)
Metric	`PLANNED-S`, `STARTED-S`, `COMPLETED-S`, `DELAY-S`, `LATE-S`, `DURATION-S`

All time values are **seconds-from-midnight**; `DELAY-S`/`LATE-S`/`DURATION-S` are durations in seconds. The generated alarm watches `LATE-S` (OK/WARNING/CRITICAL/FATAL bands from the event's `Alarm` block); it produces a sister `SynchroEvents-ALARM` series exactly like other alarms.

Event definition object

```
{
  "UniqueID": "RPT-DAILY-SALES",
  "Active": true,
```

```
  "Name": "Daily Sales Report",
  "Type": "Report",
  "Schedule": "0 8 * * 1-5",
  "AcceptableDelay": 300,
  "Dependency": "RPT-EXTRACT",
  "Offset": 0,
  "TriggeringEvent": null,
  "Alarm": { "Active": true, "WarnSec": 0, "CriticalSec": 300, "FatalSec":
900 },
  "Comments": "runs after the extract completes"
}
```

Field	Meaning
`UniqueID`	Stable id; also the file name and the metric Component.
`Active`	When false the event is not monitored (no metrics, no alarm).
`Name` / `Type`	Display name; type becomes the metric Target.
`Schedule`	Standard 5-field CRON (`min hour dom month dow`) → today's planned time.
`AcceptableDelay`	Seconds after the planned time still tolerated. `deadline = planned + AcceptableDelay`.
`Dependency` / `Offset`	Optional predecessor `UniqueID`; the dependent event's deadline is `predecessorCompletion + Offset`.
`Alarm.WarnSec/CriticalSec/FatalSec`	Lateness (seconds past deadline) at which the level escalates.

GET /events

List all event definitions (any authenticated user).

```
curl http://localhost:8080/api/v1/events -H "Authorization: Bearer $TOKEN"
```

Response: `{ "events": [ <event definition>, … ] }`

GET /events/{id}

Return one definition (404 if unknown).

```
curl http://localhost:8080/api/v1/events/RPT-DAILY-SALES -H "Authorization:
Bearer $TOKEN"
```

POST /events/{id}

Create or update a definition (**admin only**). The `UniqueID` in the body must match the URL id. Writes `data/Events/{id}.json`, regenerates the event's alarm file, and hot-reloads the monitor + alarm manager.

```
curl -X POST http://localhost:8080/api/v1/events/RPT-DAILY-SALES \
  -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"UniqueID":"RPT-DAILY-SALES","Active":true,"Name":"Daily Sales Report",
       "Type":"Report","Schedule":"0 8 * * 1-5","AcceptableDelay":300,
       "Alarm":{"Active":true,"WarnSec":0,"CriticalSec":300,"FatalSec":900}}'
```

Response: `{"UniqueID":"RPT-DAILY-SALES","status":"saved"}`

DELETE /events/{id}

Delete a definition and its generated alarm file (**admin only**).

```
curl -X DELETE http://localhost:8080/api/v1/events/RPT-DAILY-SALES \
  -H "Authorization: Bearer $TOKEN"
```

GET /events/state

Live runtime state for **all** events (active and inactive, planned-today and not) — the dashboard feed.

```
curl http://localhost:8080/api/v1/events/state -H "Authorization: Bearer $TOKEN"
```

Response:


```
{
  "events": [
    {
      "UniqueID": "RPT-DAILY-SALES",
      "Name": "Daily Sales Report",
      "Type": "Report",
      "Active": true,
      "PlannedToday": true,
      "Schedule": "0 8 * * 1-5",
      "Dependency": null,
      "Status": "DELAYED",
      "AlarmLevel": 0,
      "PlannedTime": 28800,
      "StartedTime": null,
      "CompletedTime": null,
      "ActualDelay": 0,
      "ActualLate": 0,
      "EstimatedCompletion": 29100,
```

```
      "TimeToAlarm": 120,
      "Comments": "",
      "AcceptableDelay": 300
    }
  ],
  "serverTime": 28850
}
```

`Status` is one of `SCHEDULED | STARTED | COMPLETED | DELAYED | LATE`. `AlarmLevel` is `0=OK 1=Warning 2=Critical 3=Fatal`. Time fields are seconds-from-midnight (or `null`). `EstimatedCompletion` uses the historical average of `DURATION-S`.

POST /events/{id}/start

Signal that the event actually started (any authenticated user — e.g. a `curl` line in a cron job or app hook). Body is optional.

```
curl -X POST http://localhost:8080/api/v1/events/RPT-DAILY-SALES/start \
  -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"comments":"nightly run begun"}'
```

Body fields (all optional): `time` (seconds-from-midnight; defaults to now), `comments`. Response: `{"UniqueID":"…","status":"started"}` (404 if unknown).

POST /events/{id}/complete

Signal completion. Records the completion time, success flag and comments, and freezes `DURATION-S/LATE-S`.

```
curl -X POST http://localhost:8080/api/v1/events/RPT-DAILY-SALES/complete \
  -H "Authorization: Bearer $TOKEN" -H 'Content-Type: application/json' \
  -d '{"success":true,"comments":"3.1M rows"}'
```

Body fields (all optional): `time`, `success` (default true), `comments`. Response: `{"UniqueID":"…","status":"completed"}`.

Error Responses

All errors follow the format:

```
{
  "error": "description of the error",
  "code": 401
}
```

Common error codes: | Code | Meaning | |——|———| | 400 | Bad request (missing/invalid parameters) | | 401 | Missing or invalid Authorization header | | 403 | Access denied (insufficient permissions) | | 404 | Resource not found | | 405 | Method not allowed | | 500 | Internal server error |

CORS

All endpoints include CORS headers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT, DELETE, OPTIONS
Access-Control-Allow-Headers: Authorization, Content-Type
```

Quick Start Script

```bash
#!/bin/bash
HOST="localhost:8080"
API="http://$HOST/api/v1"

# Login
TOKEN=$(curl -s -X POST "$API/auth/login" \
  -H 'Content-Type: application/json' \
  -d '{"username":"synchro","password":"synchro"}' | jq -r '.token')

echo "Token: ${TOKEN:0:20}..."

# Health
echo "=== Health ==="
curl -s "$API/health" | jq .

# Index
echo "=== Index ==="
curl -s "$API/index" -H "Authorization: Bearer $TOKEN" | jq '.ts_count'

# Query CPU
echo "=== CPU Last ==="
curl -s "$API/data/query" -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d 'function=LAST' -d 'limit=1' | jq '.data.ts[0].last'

# Timeseries (last hour)
echo "=== CPU Timeseries ==="
NOW=$(date +%s)
curl -s "$API/data/timeseries" -H "Authorization: Bearer $TOKEN" \
  -G -d 'component=affa5' -d 'type=HOST' -d 'metric=CPU' -d 'target=HOST' \
  -d "from=$((NOW-3600))" -d "to=$NOW" -d 'max_points=60' | jq '.point_count'
```